PROBABILISTIC DIVIDE-AND-CONQUER: A NEW EXACT SIMULATION METHOD, WITH INTEGER PARTITIONS AS AN EXAMPLE

RICHARD ARRATIA AND STEPHEN DESALVO

Abstract. Many natural simulation tasks, such as generating a partition of a large integer n, choosing uniformly over the p_n conceivable choices, might be done by proposing a random object from a larger set, and then checking for membership in the desired target set. The success probability, i.e., the probability that a proposed random object hits the target, may be very small.

The usual method of rare event simulation, the exponential shift or Cramér twist, is analogous to the saddle point method, and can remove the exponentially decaying part of the success probability, but even after this, the success probability may still be so small as to be an obstacle to simulation.

To simulate random integer partitions of n, using Fristedt's method, the initial proposal is a partition of a random integer of size around n, so that the counts of parts of each size are mutually independent. The usual method corresponds to evaluating the generating function at $\exp(-\pi/\sqrt{6n})$, and the remaining small probability is asymptotic to $(96n^3)^{-1/4}$.

Here, we propose a new method, probabilistic divide-and-conquer, for dealing with the small probability, e.g., the order $n^{-3/4}$ probability in the example of integer partitions. This method is analogous to changing a very difficult game, in which "hole in one" is the only way to score, to the usual game of golf.

There are many variations on the basic idea, including a simulation technique we call mix-and-match, with features of the coupon collector's problem.

For the case of integer partitions, we have a close to ideal recursive scheme, not involving mix-and-match. The asymptotic cost is $\sqrt{2}$ times the cost to propose a random partition of a random integer of size around n, so that the algorithm is within $O(1)$ of the entropy lower bound.

Contents

Date: October 17, 2011, 4:30pm version submitted to CPC. 3 typos corrected 10/19.

1. Introduction.

1.1. **Exact simulation.** Exact simulation methods provide samples from a set of objects according to some given probability distribution. For many combinatorial problems, the given distribution is the uniform choice over all possibilities of a given size.

An important technique from von Neumann 1951, [32], which we review in Section 2.2, is acceptance/rejection sampling, where one samples from an easy-to-simulate distribution related to the desired distribution, and then rejects some samples. The precise recipe is to accept samples with probability proportional to the ratio of the desired probability to the probability under the easy-to-simulate distribution. The result is that upon acceptance, the sample is an exact sample from the desired distribution.

Another important technique is Markov chain coupling from the past (CFTP), where one keeps track of coalescing Markov chains starting from some time in the past, and constructs a long enough past that all chains have coalesced by time zero [30]. This is now a very lively subject, with over 160 papers described at
`http://dimacs.rutgers.edu/∼dbwilson/exact.html/`

Divide-and-conquer is a basic strategy for algorithms. Notable examples include the Cooley-Tukey fast Fourier transform, attributable to Gauss [15], and Karatsuba's fast multiplication algorithm [16], which surprised Kolmogorov, [17]. We note that these and other cases treated in textbooks on algorithms are *deterministic*. Randomized quicksort, see for example [8] Section 7.3, is the prototype of divide-and-conquer using randomness, but such algorithms can be thought of as a variation on the deterministic algorithm, applied to a permutation of the input data.

In this paper, we propose a new method for exact sampling: probabilistic divide-and-conquer; here, the subdivision of the original problem is inherently random. We prove, in a couple of general settings, that this method does achieve exact samples.

Then we illustrate the use of this general method, with the target being to sample, for a given n, uniformly from the p_n partitions λ of the integer n. The starting point is Fristedt's construction, [12], with a random integer T of size around n, such that, for a random partition of T, the counts of parts of various sizes are mutually independent; we review this in Section 4.2. The problem then is how to simulate efficiently, since the event $T = n$ is a rare event, as in [7].

In order of ontogeny these methods express a partition as $\lambda = (A, B)$, where

(1) A is the (list of) large parts, say $\sqrt{n}$ up to n, B is the small parts, size 1 up to $\sqrt{n}$. Mix-and-match may offer an additional speedup.
(2) B is the number of parts of size 1, A is everything else. Hence the B side of the simulation is trivial, with no calls to a random number generator. Nevertheless, there is a large speedup, by a factor asymptotic to $\sqrt{n}/c$, where $c = \pi/\sqrt{6}$.
(3) In

$$p(z) = d(z)p(z^2),$$

with the classical $p(z) = \sum_{n\geq 0} p_n z^n = \prod_{i\geq 1}(1-z^i)^{-1}$, and $d(z) = \prod_{i\geq 1}(1+z^i)$ to enumerate partitions with distinct parts, A corresponds to $d(z)$. This method iterates beautifully, reducing the target, n, by a factor of approximately 4 per iteration, with an acceptance/rejection cost of only roughly $2\sqrt{2}$, improved to $\sqrt{2}$ in Section 4.5.1. We have run this on a personal computer, with n as large as 2^{49}, and relative to the basic algorithm "waiting-to-get-lucky", analyzed in Section 4.2, this version of divide-and-conquer achieves roughly a billion-fold speedup.[1]

1.2. **Probabilistic divide-and-conquer, motivated by mix-and-match.** For us, the random objects S whose simulation might benefit from a divide-and-conquer approach are those that can be described as $S = (A, B)$, where there is some ability to simulate A and B separately. Specifically, we require that $A \in \mathcal{A}$, $B \in \mathcal{B}$, and that there is a function $h : \mathcal{A} \times \mathcal{B} \to \{0, 1\}$, so that for $a \in \mathcal{A}, b \in \mathcal{B}$, $h(a, b)$ is the indicator that "a and b fit together to form an acceptable s." Furthermore, we require that A and B be independent, and that the desired S be equal in distribution to $(\ (A, B)|h(A, B) = 1)$. This description, independent objects conditional on some event, may seem restrictive, but [5, 11] shows that very broad classes — combinatorial assemblies, multisets, and selections — fit into this setup.

Now imagine that one wants an honest sample of size m, that is, $S_1, S_2, \ldots, S_m$, to be independent, with the original S along with $S_1, S_2, \ldots$ to be identically distributed. The pedestrian approach is to propose sample values $A_1, A_2, \ldots$, and $B_1, B_2, \ldots$, and to consider the indicators of *aligned* matching, that is, $h(A_1, B_1), h(A_2, B_2), \ldots$. One naturally has waiting times $T_1, T_2, \ldots$, with $T_1 := \min\{i \geq 1 : h(A_i, B_i) = 1\}$, and for for $k > 1$, $T_k := \min\{i > T_{k-1} : h(A_i, B_i) = 1\}$. And of course, the ℓth sample found is $S_\ell := (A_{T_\ell}, B_{T_\ell})$.

In the case where $k = T_m$ is large, the scheme just described seems wasteful — we proposed k values of $A \in \mathcal{A}$, and k values of $B \in \mathcal{B}$, and hence might have k^2 opportunities for a match. That is, rather than just look for the aligned potential matches, scored by the $h(A_i, B_i)$ for $1 \leq i \leq k$, we might have considered the $h(A_i, B_j)$ for $1 \leq i, j \leq k$, with k^2 index pairs (i, j). Indeed, this situation arises naturally in biological sequence matching, [6, 4], where for two independent sequences

[1]The RandomPartition function in Mathematica® [23] appears to hit the wall at around $n = 2^{20}$.

of iid letters, in many *but not all* cases for the two marginal distributions, unrestricted rather than aligned matching effectively squares the number of ways to look for a match, and hence approximately doubles the length of the longest match found.

Of course, the difficulty with the general program "search all k^2 index pairs ij" is that conflicting matches might be found. Suppose, for example, that exactly two matches are found, with A_i matching both B_j and $B_{j'}$, with $j \neq j'$. It is easy to see that taking both AB pairs ruins the iid nature of a sample. Also, though perhaps not as obvious, other strategies, such as suppressing both AB pairs, or taking only the pair indexed by the lexicographically first of ij, ij', or taking only one pair, based on an additional coin toss, introduce bias relative to the desired distribution for S.

There is a way to allow mix-and-match, and still get an honest sample[2]; details will be given in Section 3. The first step is to use rejection sampling to produce a list of As, distributed as a sample from *the distribution of As biased by the chance that they would match, if a single B were proposed.* For us, it only became apparent later, that this use of rejection, even in the absence of mix-and-match, can be useful; Theorem 2 describes a surprising example.

2. The basic lemma for exact sampling with divide-and-conquer

We assume throughout that

$$A \in \mathcal{A},\ B \in \mathcal{B} \text{ have given distributions,} \tag{1}$$

$$A, B \text{ are independent,} \tag{2}$$

$$h : \mathcal{A} \times \mathcal{B} \to \{0,1\} \tag{3}$$

$$\text{satisfies } p := \mathbb{E}\, h(A,B) \in (0,1],^{3}$$

where, of course, we also assume that h is measurable, and

$$S \in \mathcal{A} \times \mathcal{B} \text{ has distribution } \mathcal{L}(S) = \mathcal{L}(\,(A,B) \mid h(A,B) = 1), \tag{4}$$

i.e., the law of S is the law of the independent pair (A,B) *conditional on* having $h(A,B) = 1$.

[2]under a condition on the structure of h, described by Lemma 2.

[3] The requirement that $p > 0$ is *not needed* for divide-and-conquer to be useful; but rather, a choice we make for the sake of simpler exposition. In cases where $p = 0$, the conditional distribution, apparently specified by (4), needs further specification — this is known as Borel's paradox.

Note that a restatement of (4), exploiting the hypothesis (3), is that for measurable sets $R \subset \mathcal{A} \times \mathcal{B}$,

$$\mathbb{P}(S \in R) = \frac{\mathbb{P}((A,B) \in R \text{ and } h(A,B) = 1)}{p},$$

or equivalently, for bounded measurable functions g from $\mathcal{A} \times \mathcal{B}$ to the real numbers,

$$\mathbb{E}\, g(S) = \mathbb{E}\left(g(A,B)h(A,B)\right)/p.$$

Since we have assumed $p > 0$, this is elementary conditioning. This allows the distributions of A and B to be arbitrary: discrete, absolutely continuous, or otherwise.

The following lemma is a straightforward application of Bayes' formula.

Lemma 1. *Suppose X is the random element of $\mathcal{A}$ with distribution*

$$\mathcal{L}(X) = \mathcal{L}(\, A \,|\, h(A,B) = 1\,), \tag{5}$$

and Y is the random element of $\mathcal{B}$ with conditional distribution

$$\mathcal{L}(Y\,|X = a) = \mathcal{L}(\, B \,|\, h(a,B) = 1\,). \tag{6}$$

Then $(X,Y) =^d S$, i.e. the pair (X,Y) has the same distribution as S, given by (4).

Proof. A restatement of (5) is

$$\mathbb{P}(X \in da) = \frac{\mathbb{P}(A \in da)\, Eh(a,B)}{p}, \tag{7}$$

and relation (6) is equivalent to

$$\mathcal{L}((X,Y)\,|X = a) = \mathcal{L}(\,(a,B)\,|\, h(a,B) = 1\,). \tag{8}$$

Hence, for any bounded measurable $g : \mathcal{A} \times \mathcal{B} \to \mathbb{R}$,

$$\begin{aligned}
\mathbb{E}\, g(X,Y) &= \mathbb{E}\left(\mathbb{E}\left(g(X,Y)|X\right)\right) \\
&= \int_{\mathcal{A}} \mathbb{P}(X \in da) \quad \mathbb{E}\left(g(X,Y)|X = a\right) \\
&= \int_{\mathcal{A}} \frac{\mathbb{P}(A \in da)\,\mathbb{E}\, h(a,B)}{p}\ \frac{\mathbb{E}\left(g(a,B)h(a,B)\right)}{\mathbb{E}\, h(a,B)} \\
&= \frac{1}{p}\int_{\mathcal{A}} \mathbb{P}(A \in da)\ \mathbb{E}\left(g(a,B)h(a,B)\right) \\
&= \mathbb{E}\left(g(A,B)|h(A,B) = 1\right) = \mathbb{E}\, g(S).
\end{aligned}$$

We used (8) for the middle line in the display above; on the set $\mathcal{A}_0 := \{a \in \mathcal{A} : \mathbb{E}\, h(a,B) = 0\}$, which contributes 0 to the integral, we took the usual liberty of dividing by 0, rather than writing out separate expressions for the integrals over $\mathcal{A}_0$ and $\mathcal{A} \setminus \mathcal{A}_0$.

□

2.1. **Algorithmic implications of the basic lemma.** Assume that one wants a sample of fixed size m from the distribution of S. That is, one wants to carry out a simulation that provides $S_1, S_2, \ldots$, with $S_1, S_2, \ldots, S_m$ being mutually independent, with each equal in distribution to S. According to Lemma 1, this can be done by providing m independent pairs (X_i, Y_i), $i = 1$ to m, each equal in distribution to S.

A reasonable choice of how to carry this out, not yet using mix-and-match, involves the following:

Outline of an algorithm to gather sample of size m.
Stage 1. Sample $X_1, X_2, \ldots, X_m$ from the distribution of X, i.e., from $\mathcal{L}(X) = \mathcal{L}(\, A \,|\, h(A,B) = 1\,)$.
Stage 2. Conditional, on the result of stage 1 producing $(X_1, \ldots, X_m) = (a_1, \ldots, a_m)$, find $Y_1, Y_2, \ldots, Y_m$, mutually independent, with distributions $\mathcal{L}(Y_i) = \mathcal{L}(\, B \,|\, h(a_i, B) = 1\,)$.

Note that in general, conditional on the result of stage 1, the Y_i in stage 2 are *not identically* distributed. Furthermore, the trials undertaken to find these Y_i need not be independent of each other.

2.2. **Use of acceptance/rejection sampling.** Assume that we know how to simulate A — this is under the distribution in (1), where A, B are independent. But, we need instead to sample from an alternate distribution, denoted above as that of $X \in \mathcal{A}$. The rejection method recipe, for using (7), may be viewed as having 4 steps, as follows.

(1) Find a threshold function $t : \mathcal{A} \to [0,1]$, with $t(a)$ proportional to $\mathbb{E}\, h(a,B)/p$, i.e., t of the form $t(a) = C \times \mathbb{E}\, h(a,B)/p$ for some positive constant C.
(2) Propose iid samples $A_1, A_2, \ldots$,
(3) Independently generate uniform (0,1) random variables $U_1, U_2, \ldots$,
(4) If $U_i \le t(A_i)$, then accept A_i as as X value; otherwise reject it.

The cost of using acceptance/rejection.
Naturally, one wants to take the constant C for the threshold function t in step (1) to be as large as possible. This is subject to the constraint $t(a) \le 1$ for all a, i.e., $C \times \mathbb{E}\, h(a,B)/p \le 1$. The expected fraction of proposed samples A_i to be accepted will be the average of $t(a)$ with

respect to the distribution of A, i.e.,

$$p_{\text{acc}} := \mathbb{P}(U \leq t(A)) = \mathbb{E}\, t(A) = \mathbb{E}\left(C \times \frac{h(A,B)}{p}\right) = C,$$

and the expected number of proposals needed to get each acceptance is the reciprocal of this, so we define

$$\text{Acceptance cost} := \frac{1}{p_{\text{acc}}} = \frac{1}{C}. \tag{9}$$

Assuming that we can find an a^* where $\mathbb{E}\, h(a,B)$ achieves its maximum value, this simplifies to

$$\text{Acceptance cost} = \frac{1}{C} = \frac{p}{\mathbb{E}\, h(a^*,B)}, \tag{10}$$

For comparison, if we were not using divide-and-conquer, but instead proposing pairs (A_i, B_i) and hoping to get lucky, i.e., hoping that $h(A_i, B_i) = 1$, with success probability p and expected number of proposals to get one success equal to $1/p$, then, *ignoring the cost of proposing the* B_i, the **speedup** involved in (10) is a factor of $1/\mathbb{E}\, h(a^*, B)$.

Is the threshold function t computable? In step (4), for each proposed value $a = A_i$, we need to be able to compute $t(a)$; this can be either a minor cost, a major cost, or an absolute impediment, making probabilistic divide-and-conquer infeasible. All of these variations occur, in the context of integer partitions, and will be discussed in Sections 4.3 – 4.5, and again in Section 5.2.

2.3. **Caution: mix-and-match not yet enabled.**

In Stage 2 of the 2-stage algorithm described at the beginning of Section 2.1, there is a subtle issue involved in the phrase "independently find $Y_1, Y_2, \ldots, Y_m$ $\cdots$". Suppose, for example, that one plans to propose iid copies of B, say $B_1, B_2, \ldots$, waiting for samples that will match one of the $a_1, a_2, \ldots, a_m$ obtained in the first stage.

A correct way to carry out stage 2 is to consider stopping times $1 \leq \alpha_1 < \alpha_2 < \cdots < \alpha_m$, with

$$\alpha_1 := \min\{j \geq 1 \colon h(a_1, B_j) = 1\}, \tag{11}$$

and then recursively, for $\ell = 2$ to m, $\alpha_\ell := \min\{j \geq \alpha_{\ell-1} \colon h(a_\ell, B_j) = 1\}$. Here, stage 2 is completed after proposing α_m copies of B, and the m samples of S are $(a_\ell, B_{\alpha_\ell})$ for $\ell = 1$ to m.

In general, it would be incorrect to try to speed up stage 2 by taking stopping times $1 \leq \beta_1 < \beta_2 < \cdots < \beta_m$, with

$$\beta_1 := \min\{j \geq 1 \colon h(a_i, B_j) = 1 \text{ for some } i \in \{1, 2, \ldots, m\}\}, \tag{12}$$

and then $\beta_2 := \min\{j \geq \beta_1 \colon h(a_i, B_j) = 1$ for one of the $m-1$ values i not already matched$\}$, and so on. [In case the B at time β_1 matches a_i for more than one index i, we might choose, for example, to declare the smallest available i to serve as the index matched, for the sake of specifying which $m-1$ indices are available in the definition of β_2. Other choices as to which i, even those involving auxiliary randomization, will have the same effect.] Say that $I(\ell)$ is the index matched at time β_ℓ. With the random permutation π defined to be the inverse of the permutation $I(\cdot)$, the sample consists of $(a_\ell, B_{\beta_{\pi(\ell)}})$ for $\ell = 1$ to m. Not only there may be *dependence* in the sequence $B_{\beta_{\pi(1)}}, B_{\beta_{\pi(2)}}, \ldots, B_{\beta_{\pi(m)}}$, it may also be the case[4] that $B_{\beta_{\pi(\ell)}}$ *fails* to have the desired marginal distribution, i.e. that of B conditional on $h(a_\ell, B) = 1$. (Perhaps this B was matched to a_ℓ when a higher priority index i was available; the fact of not satisfying $h(a_i, B) = 1$ biases the distribution.)

3. *Simple* matching enables mix-and-match

Lemma 1 was basic, and so is the following lemma, but the pair serves nicely to clarify the logical structure of what is needed to enable probabilistic divide-and-conquer, versus what is needed to further enable mix-and-match.

Lemma 2. *Given $h : \mathcal{A} \times \mathcal{B} \to \{0,1\}$, the following two conditions are equivalent:*
Condition 1: $\forall a, a' \in \mathcal{A}, b, b' \in \mathcal{B}$,

$$1 = h(a,b) = h(a',b) = h(a,b') \text{ implies } h(a',b') = 1, \tag{13}$$

Condition 2: $\exists \mathcal{C}$, and functions $c_\mathcal{A} : \mathcal{A} \to \mathcal{C}, c_\mathcal{B} : \mathcal{B} \to \mathcal{C}$, so that $\forall (a,b) \in \mathcal{A} \times \mathcal{B}$,

$$h(a,b) = 1(c_\mathcal{A}(a) = c_\mathcal{B}(b)). \tag{14}$$

We think of $\mathcal{C}$ as a set of colors*, so that condition* (14) *says that a and b match if and only if they have the same color.*

Proof. That (14) implies (13) is trivial. In the other direction, it is easy to check that (13) implies that the relation $\sim_\mathcal{A}$ on $\mathcal{A}$ given by $a \sim_\mathcal{A} a'$ iff $\exists b \in \mathcal{B}, 1 = h(a,b) = h(a',b)$ is an *equivalence* relation. Likewise for the relation $\sim_\mathcal{B}$ on $\mathcal{B}$ given by $b \sim_\mathcal{B} b'$ iff $\exists a \in \mathcal{A}, 1 = h(a,b) = h(a,b')$. For the set of colors, $\mathcal{C}$, we might take either the set of equivalence classes of $\mathcal{A}$ modulo $\sim_\mathcal{A}$, or the set of equivalence classes of $\mathcal{B}$ modulo $\sim_\mathcal{B}$, and then (13) also provides a bijection between these two sets of equivalence classes, to induce (14). $\square$

[4]We thank Sheldon Ross for first observing this.

Remark 1. *The proof of Lemma 2, with equivalence classes of $\mathcal{A}/{\sim_\mathcal{A}}$ and $\mathcal{B}/{\sim_\mathcal{B}}$, shows that the pair of coloring functions satisfying* (14) *is* essentially unique*. Specifically, unique apart from relabeling and padding, i.e., an arbitrary permutation on the names of the colors used, and enlarging the range, $\mathcal{C}$, to an arbitrary superset of the image.*

Remark 2. *The statement of Lemma 2 shows that coloring is* not essentially *an issue of* sufficient statistics*. After all, hypothesis* (13) *only concerns the logical structure of the matching function h appearing in* (3)*, and doesn't involve the distributions on $\mathcal{A}$ and $\mathcal{B}$ appearing in* (1)*.*

Remark 3. *When* (14) *holds, we can write the event that A matches B as a union indexed by the color involved:*

$$\{h(A,B)=1\} = \cup_{k\in\mathcal{C}}\{c_\mathcal{A}(A)=k, c_\mathcal{B}(B)=k\},$$

so that $p=\sum_{k\in\mathcal{C}}\mathbb{P}(c_\mathcal{A}(A)=k, c_\mathcal{B}(B)=k)$, and we see that at most a countable set of colors k contribute a strictly positive amount to p. As a notational convenience, we take $\mathbb{N}\subset\mathcal{C}$, and use positive integers k for the names of colors that have

$$\mathbb{P}(c_\mathcal{A}(A)=k, c_B(B)=k) > 0. \tag{15}$$

[The fact that A,B are independent, hence $\mathbb{P}(c_\mathcal{A}(A)=k, c_B(B)=k) = \mathbb{P}(c_\mathcal{A}(A)=k)\mathbb{P}(c_B(B)=k)$, is irrelevant to main idea behind (15)*. However, a technique we refer to as 'roaming x' in Section 4.3.2 is an example of how to take advantage of the independence of A and B.]*

The intent of the following lemma is to show that if h satisfies (14), then mix-and-match strategies can can be used in stage 2 of the broad outline of Section 2.1.

Lemma 3. *Assume that h satisfies* (14)*. Consider a procedure which proposes a sequence $D_1, D_2, \ldots$ of elements of $\mathcal{B}$ with the following properties:*

There is a sequence of $\sigma-$algebras $\mathcal{F}_0\subset\mathcal{F}_1\subset\mathcal{F}_2\subset\cdots$. [We think of $\mathcal{F}_0$ as carrying the information from stage 1 of an algorithm along the lines described in Section 2.1, carrying information such as "which demands $a_1, a_2, \ldots$ must be met — or reduced information, such as the colors $c_\mathcal{A}(a_1),\ldots,c_\mathcal{A}(a_m)$.]

For every $n\ge 1$ and k satisying (15)*, conditional on $\mathcal{F}_{n-1}$ together with $c_\mathcal{B}(D_n)=k$, the distribution of D_n is equal to $\mathcal{L}(B|c_\mathcal{B}(B)=k)$.*

For every k satisying (15)*,*

$$\text{with probability 1, infinitely many } n \text{ have } c_\mathcal{B}(D_n)=k. \tag{16}$$

Define stopping times $\tau_i^{(k)}$, the "time n of the i-th instance of $c_{\mathcal{B}}(D_n) = k$, by $\tau_0^{(k)} = 0$ and for $i \geq 1, \tau_i^{(k)} = \inf\{n > \tau_{i-1}^{(k)} : c_{\mathcal{B}}(D_n) = k\}$. We write $D(n) \equiv D_n$, to avoid multi-level subscripting, and define $B_i^{(k)} := D(\tau_i^{(k)})$ for $i = 1, 2, \ldots$.

Then*, for each k, the $B_1^{(k)}, B_2^{(k)}, \ldots$ are independent, with the distribution $\mathcal{L}(B|c_{\mathcal{B}}(B) = k)$, and as k varies, these sequences are mutually independent.*

Proof. The proof is a routine exercise; it suffices to check the independence claim for an arbitrary finite number of choices of k, restricting to the $B_i^{(k)}$ for $1 \leq i \leq i_0$ with an arbitrary finite i_0, and this can be done, along with checking for the specified marginal distributions, by summing over all possible values for the random times $\tau_i^{(k)}$. Writing out the full argument would be notationally messy, and not interesting. □

4. Algorithms for simulating random partitions of n

In this section we focus on the generation of partitions under the uniform distribution. Other distributions on partitions such as the Plancherel measure, while important in many applications, for example [10, 13], are not addressed in the current paper, but it is highly plausible that these techniques can be adapted to those measures. We note that the phrase "generating partitions" is usually taken, as in [19], to mean the systematic listing of all partitions, perhaps subject to additional constraints; this is very different from simulating a random instance.

The computational analysis that follows uses an informal adaptation of *uniform* costing; see Section 5.2. Some elements of the analysis, specifically asymptotics for the acceptance rate, will be given rigorously, for example in Theorems 1, 2, and 3.

4.1. For baseline comparison: table methods.

A natural simulation method is to find the largest part first, then the second largest part, and so on according to a distribution of largest part[5]. The main cost associated with this method is the storage of all the distributions needed. Some details follow.

Let $p(\leq k, n)$ denote the number of partitions of n with each part of size $\leq k$, so that $p(\leq n, n) = p_n$. These can be quickly calculated from the recurrence

$$p(\leq k, n) = p(\leq k-1, n) + p(\leq k, n-k), \tag{17}$$

[5]For the largest part, we are are dealing with *unrestricted* partitions of n, but for subsequent parts, the problem involves the largest part of a partition of an integer m into parts of size at most k.

where the right hand side counts the number of partitions without any k's plus the number of partitions with at least one k.

Let X_i denote the i-th largest part of a randomly generated partition, so $\lambda = (X_1, X_2, \ldots)$. We have

$$\mathbb{P}(X_1 \le j) = p(\le j, n)/p_n,$$

$$\mathbb{P}(X_2 \le s | X_1 = j) = p(\le s, n-j)/p(\le j, n-j),$$

and in general for $i \ge 2$

$$\mathbb{P}(X_i \le j_i | X_1 = j_1, \ldots, X_{i-1} = j_{i-1}) = p(\le j_i, n - \sum j_\ell)/p(\le j_{i-1}, n - \sum j_\ell).$$

Rather than computing each quantity on the right hand side as it appears, an n by n table, where the j-th column consists of the numbers $p(\le 1, j), p(\le 2, j), \ldots, p(\le n, j)$, can be computed and stored, hence generating partitions is extremely fast once this table has been created. For one uniformly distributed random number, we can look up the value of the largest part of a random partition; this implies order of $\sqrt{n}\,\log n$ lookups. An easy variation also finds the multiplicity of the largest part, implying order of $\sqrt{n}$ lookups.

The memory conditions are on the order of n^2 for the entire table, a severe constraint, but once it has been constructed the generation of random partitions is rapid.[6] Another variation on this method, based on Euler's identity $np_n = \sum_{d,j\ge 1} dp_{n-dj}$, is given in [25, 26], and cited as "the recursive method." We haven't found a clearcut complexity analysis in the literature, although [9] comes close. But we believe that this algorithm is less useful than the $p(\le k, n)$ table method — sampling from the distribution on (d, j) implicit in $np_n = \sum_{d,j\ge 1} dp_{n-dj}$ requires computation of partial sums; if all the partial sums for $\sum_{d,j\ge 1, dj\le m} dp_{m-dj}$ with $m \le n$ are stored, the total storage requirement is of order $n^2 \log n$, and if they aren't stored, computing the values needed, on the fly, becomes a bottleneck.

[6]Since the largest part of a random partition is extremely unlikely to exceed $O(\sqrt{n}\log n)$, one can get away using a table of size $O(n^{3/2}\log n)$, which will only rarely need to be augmented. Specifically, writing λ_1' for the largest part of a partition, with $c = \pi/\sqrt{6}$, for fixed $A > 0$, with $i_0 = i_0(n, A) := \sqrt{n}\log(A\sqrt{n}/c)/c$, $\mathbb{P}_n(\lambda_1' \ge i_0) \to 1 - \exp(-1/A)$. For example, take $A = 1000$, so that the i_0 by n table will need to be augmented with probability approximately .001. With M words of memory available for the table, we solve $i_0(n, A) \times n = M$; for example, with $M = 2^{28}$ and $A = 1000$ we have $n = 15000 \doteq 2^{17}$ and $i_0 = 1700$, and increasing M to 2^{37} gets us up to $n = 9 \times 10^6 \doteq 2^{23}$, $i_0 = 15000$. Instead of actually augmenting the table, one could treat the roughly one out of every A computationally difficult cases as *deliberately missing data*, as in Section 4.3.3.

4.2. **Waiting to get lucky.** Hardy and Ramanujan [14] proved the asymptotic formula, as $n \to \infty$,

$$p_n \sim \exp(2c\sqrt{n})/(4\sqrt{3}n), \quad \text{where } c = \pi/\sqrt{6} \doteq 1.282550. \tag{18}$$

Fristedt [12] observed that, for any choice $x \in (0,1)$, if $Z_i \equiv Z_i(x)$ has the geometric distribution given by

$$\mathbb{P}(Z_i = k) \equiv \mathbb{P}_x(Z_i = k) = (1 - x^i)\ (x^i)^k,\ k = 0, 1, 2, \ldots, \tag{19}$$

with $Z_1, Z_2, \ldots$ mutually independent, and T is defined by

$$T = Z_1 + 2Z_2 + \cdots, \tag{20}$$

then, conditional on the event $(T = n)$, the partition λ having Z_i parts of size i, for $i = 1, 2, \ldots$, is uniformly distributed over the p_n possible partitions of n.[7]

This extremely useful observation is easily seen to be true, since for any nonnegative integers $(c(1), c(2), \ldots)$ with $c(1) + 2c(2) + \cdots = n$, specifying a partition λ of the integer n,

$$\mathbb{P}(Z_i = c_i, i = 1, 2, \ldots) = \prod \mathbb{P}(Z_i = c(i)) = \prod \left((1 - x^i)(x^i)^{c(i)}\right)$$

$$= \ x^{c(1)+2c(2)+\cdots} \prod (1 - x^i) = x^n \ \prod (1 - x^i), \tag{21}$$

which does not vary with the partition λ of n.

The event $(T = n)$ is the disjoint union, over all partitions λ of n, of the events whose probabilities are given in (21), showing that

$$\mathbb{P}_x(T = n) = p_n\ x^n\ \prod (1 - x^i). \tag{22}$$

If we are interested in random partitions of n, an especially effective choice for x, used by [12, 28], is

$$x(n) = \exp(-c/\sqrt{n}), \quad \text{where } c = \pi/\sqrt{6}. \tag{23}$$

Under this choice, we have, as $n \to \infty$,

$$\frac{1}{n}\ \mathbb{E}_{\,x(n)} T \to 1, \quad \frac{1}{n^{3/2}} \mathrm{Var}_{x(n)} T \to \frac{2}{c}; \tag{24}$$

this is essentially a pair of Riemann sums, see [5], page 106. If we write $\sigma(x)$ for the standard deviation of T, then the second part of (24) may be paraphrased as, with $x = x(n)$, as $n \to \infty$,

$$\sigma(x) \sim \sqrt{2/c}\ n^{3/4}. \tag{25}$$

[7]We write $\mathbb{P}_x$ or $\mathbb{P}$, or Z_i or $Z_i(x)$ interchangeably, depending on whether the choice of $x \in (0,1)$ needs to be emphasized, or left understood.

The *local central limit heuristic* would thus suggest asymptotics for $\mathbb{P}_x(T = n)$, and these simplify, using (23) and (24), as follows:

$$\mathbb{P}_x(T = n) \sim \frac{1}{\sqrt{2\pi}\ \sigma(x)} \sim \frac{1}{2\sqrt[4]{6}\, n^{3/4}}. \tag{26}$$

The Hardy-Ramanujan asymptotics (18) and the exact formula (22) combine to show that (26) does hold.

Theorem 1. Analysis of *Waiting-to-get-lucky.*
Consider the following algorithm to generate a random partition of n, chosen uniformly from the $p_n \sim \exp(2c\sqrt{n})/(4\sqrt{3}\, n)$ possibilities. Use the distributions in (19)*, with parameter x given by* (23)*.*

(1) *Propose a sample, $Z_1, Z_2, \ldots, Z_n$; compute $T_n := Z_1 + 2Z_2 + \cdots + nZ_n$.*
(2) *In case ($T_n = n$), we have* got lucky*. Report the partition λ with Z_i parts of size i, for $i = 1$ to n, and stop. Otherwise, repeat from the beginning.*

This algorithm does *produce one sample from the desired distribution, and the expected number of proposals until we get lucky is asymptotic to $2\sqrt[4]{6}\, n^{3/4}$.*

Proof. It is easily seen that $\mathbb{P}_x(Z_i = 0 \text{ for all } i > n) \to 1$. [In detail, $\mathbb{P}(\text{not } (Z_i = 0 \text{ for all } i > n)) \le \sum_{i>n} \mathbb{P}(Z_i \ne 0) = \sum_{i>n} x^i < \sum_{i \ge n} x^i = x^n/(1-x) \sim x^n/(c/\sqrt{n}) = \exp(-c\sqrt{n})\sqrt{n}/c \to 0$.] Hence the asymptotics (26), given for the infinite sum T, also serve for the finite sum T_n, in which the number of summands, along with the parameter $x = x(n)$, varies with n. $\square$

Remark 4. *We are* not *claiming that the running time of the algorithm grows like $n^{3/4}$, but only that the number of proposals needed to get one acceptable sample grows like $n^{3/4}$. The time to propose a sample also grows with n. Assigning cost 1 to each call to the random number generator, with all other operations being free, the cost to propose one sample grows like $\sqrt{n}$ rather than n; details in Section 5.2. Combining with Theorem 1, the cost of the waiting-to-get-lucky algorithm grows like $n^{5/4}$. A simple Matlab® program to carry out waiting-to-get lucky is presented in Section 5.1.*

4.3. **Divide-and-conquer, by small versus large.**

The waiting-to-get-lucky strategy is limited primarily by the probability that the target is hit, which diminishes like $n^{-3/4}$. Already at $n = 10^8$, the probability is one in a million. Instead of trying to hit a hole in one, we allow approach shots.

Recall that to sample partitions uniformly, based on (19) – (26), our goal is to sample from the distribution of $(Z_1, Z_2, \ldots, Z_n)$ conditional on $T = n$. Using $x = x(n)$ from (23), and with $b \in \{1, 2, \ldots, n-1\}$, we let

$$A = (Z_{b+1}, Z_{b+2}, \ldots, Z_n), \quad B = (Z_1, \ldots, Z_b). \tag{27}$$

Motivated by the standard paradigm for deterministic divide-and-conquer, that the two tasks should be roughly equal in difficulty, we will take $b = O(\sqrt{n})$, having observed that the expected largest part of a random partition grows like $\sqrt{n}\,\log n$. With

$$T_A = \sum_{i=b+1}^{n} iZ_i, \quad T_B = \sum_{i=1}^{b} iZ_i, \tag{28}$$

we want to sample from (A, B) conditional on $T_A + T_B = n$. We have A, B independent, and we use $h(A, B) = 1(T_A + T_B = n)$. The divide-and-conquer strategy, according to Lemma 1, is to sample X from the distribution of A biased by the probability that an independently chosen B will will make a match, and then, having observed $A = a$, sampling Y from the distribution of B conditional on $h(a, B) = 1$.

In order to simulate X, we will use rejection sampling, as reviewed in Section 2.2. To find the optimal rejection probabilities, we want the largest C such that

$$C \max_j \frac{\mathbb{P}(T_B = n-j)}{\mathbb{P}(T_n = n)} \leq 1,$$

or equivalently,

$$C = \frac{\mathbb{P}(T_n = n)}{\max_j \mathbb{P}(T_B = j)}. \tag{29}$$

The values $\mathbb{P}(T_B = k)$ for $k = 0, 1, \ldots, n$ can be simultaneously computed, using the recursion (17); what we really have is a variant of the n by n table method of Section 4.1, in which the table is b by n, so the computation time is $b\,n$; furthermore, one only needs to store the current and previous row, (or with overwriting, only the current row), so the storage is n. Once we have the last row of the b by n table, we can easily find C and indeed the entire threshold function t. The time to compute the table, $b\,n$, would have been the much larger $(n-b)n$ had we interchanged the roles of A and B, i.e., taken $B := (Z_{b+1}, Z_{b+2}, \ldots, Z_n)$ instead of $B := (Z_1, \ldots, Z_b)$.

In order to simulate $Y = (B | Z_1 + 2Z_2 + \cdots + bZ_b = n - k)$, in situations where b is too large to store a b by n table, we resort to waiting-to-get-lucky.[8] Our goal for the next section is to explore mix-and-match using integer partitions as an example, regardless of whether there exists a better competing algorithm for integer partitions.

4.3.1. *Divide-and-conquer with mix-and-match.* When a sample of size $m > 1$ is desired, Lemmas 1 – 3 can be used to generate unbiased samples. Observe that the matching function with $h(A, B) = 1(T_A + T_B = n)$, for $(A, B) \in \mathcal{A} \times \mathcal{B}$, satisfies condition 2 of Lemma 2, with $c_{\mathcal{A}}(A) = T_A$ and $c_{\mathcal{B}}(B) = n - T_B$. Hence *mix-and-match* is enabled. The first phase of the algorithm, phase A, generates samples $A_1, A_2, \ldots, A_m$ from X, according to Lemma 1. This creates a multiset of m colors, $\{c_1, \ldots, c_m\}$, where $c_i = c_{\mathcal{A}}(A_i)$. We think of these as m demands that must be met by phase B of the algorithm. One strategy for phase B is to generate an iid sequence of samples of B; initially, for each sample, we compute its color $c = c_{\mathcal{B}}(B)$ and check whether c is in the multiset of demands $\{c_1, \ldots, c_m\}$; when we find a match, we pair B with one of the A_i of the matching color, to produce our first sample, which we set as $S_i = (A_i, B)$. Then we reduce the multiset of demands by one element, and iterate, until all m demands have been met. Lemma 3 implies that the resulting list $(S_1, S_2, \ldots, S_m)$ is an iid sample of m values of S, as desired.

Remark 5. *Note that in the above, if the first match found, (A_i, B), is labelled as S_1, and the second matching pair is labelled S_2, and so on, then the resulting list $(S_1, S_2, \ldots, S_m)$ is* not *necessarily an iid sample of S; the colors c with $\mathbb{P}(c = c_{\mathcal{B}}(B))$ large would tend to show up earlier in the list.*

4.3.2. *Roaming x.* Consider again a sample of size $m = 1$. Having accepted $X = (Z_{b+1}, \ldots, Z_n)$ with color $k = T_A$, in the notation of (28), we now need Y, which is $B = (Z_1, \ldots, Z_b)$ conditional on having color k, which simplifies to having $n - k = T_B := \sum_{i=1}^{b} iZ_i$. One obvious strategy is to sample B repeatedly, until getting lucky. The distribution of B is specified by (19) and (23) — with a choice of parameter, $x = x(n)$, not taking into account the values of b and $n - k$. A computation similar to (21) shows that the distribution of $(Z_1(y), \ldots, Z_b(y))$ conditional on $\sum_{i=1}^{b} iZ_i(y) = n - k$ is the same, for all choices $y \in (0, 1)$.

[8]Of course, instead of trying to get lucky all at once, one might apply divide-and-conquer to this B task. But we do not pursue this particular iteration, in light of a better iteration scheme, presented in Section 4.5.

As observed in [5], the y which maximizes $\mathbb{P}_y T_B = n-k$, i.e., gives us the best chance of getting lucky, is the solution of $n-k = \sum_{i=1}^{b} \mathbb{E}\, iZ_i(y)$. Thus, in the case $m = 1$, the optimal choice of y is easily prescribed. However, for large m we also recommend using mix-and-match, which brings into play a complicated coupon collector's situation. With a multiset of demands $\{c_1, \ldots, c_m\}$ from Section 4.3.1, the algorithm designer has many choices of global strategy; it is not obvious whether or not a greedy strategy — picking y for the next proposed B, to maximize the chance that B satisfied at least one of the demands — is optimal.

4.3.3. *Deliberately missing data.* For motivation: in the B phase of mix-and-match, one situation would have all m demands $c_1, c_2, \ldots, c_m$ distinct, and in addition, $\mathbb{P}(c_B(B) = c_i)$ relatively constant as i varies. In this situation,we have the classic coupon collector's problem; with time $m \log m$ to collect all m coupons. There is a harmonic slowdown: for example, the first match is found 100 times as quickly as the match after ninetynine percent of the demands have been met; the last v matches are expected to take about $(1 + 1/2 + \cdots + 1/v)/\log m$ as long as the first $m - v$ matches combined. In the situation with the $\mathbb{P}(c_B(B) = c_i)$ relatively constant, but with large multiplicities in the multiset $\{c_1, \ldots, c_m\}$, there is an even more dramatic slowdown near the end, based on the size of the set underlying the multiset of remaining demands. Note however, the values $\mathbb{P}(c_B(B) = c_i)$ might not be relatively constant, even if we adjust the sampling distributions of B to fit the particular colors remaining to be found, as described in Section 4.3.2.

Suppose we stop before completing the B phase, with some v of demands remaining to be met. There is usable information in the partially completed sample, which has the $m - v$ partitions found so far. This list of $m - v$ is *not* a sample of size $m - v$, since *sample* requires iid from the original target distribution. But, had we run the B phase to completion, we would have had an honest sample of size m, so there is information in knowing all but v of the m values. Think of this sample of size m as *the sample*, with some v of its elements being unknown. For estimates based on the sample proportion, an error of size at most v/m is introduced by the unknown elements. Since the standard deviation, due to sample variability, decays roughly like $1/\sqrt{m}$, it makes sense to allow v comparable to $\sqrt{m}$.

For example, Pittel [29] proves that, as $n \to \infty$, given two partitions of n, choosing (λ, μ) uniformly over the $(p_n)^2$ possible pairs, the probability π_n that λ dominates μ satisfies $\pi_n \to 0$. Also, he proves that, choosing a single partition λ uniformly over the p_n possibilities, the probability π_n^* that λ dominates its dual λ' satisfies $\pi_n^* \to 0$. It is natural to guess that $\lim \pi_n/\pi_n^*$ exists, with a value in $[0, \infty]$; one can use simulations to suggest whether $0, 1$, or ∞ is the most plausible value. The harder task is to simulate for π_n.

If one has an honest sample of $2m$ partitions of n, with v missing items due to terminating the B phase early, then one would have an honest sample of $m - V$ pairs (λ, μ), with random variable $V \leq v$.[9] If we had $v = 0$, and H of the pairs have λ dominates μ, then the point estimate for $p := \pi_n$ is H/m, and the standard $(1-\alpha)\%$ confidence interval is

$$\left[\frac{H}{m} - z_{\alpha/2}\sqrt{\frac{p(1-p)}{m}}, \frac{H}{m} + z_{\alpha/2}\sqrt{\frac{p(1-p)}{m}}\right].$$

With V missing pairs, consider the count K, how many of the $m-V$ completed pairs had λ dominates μ. We can do a worst-case analysis by assuming, on one side, that all V of the missing pairs have domination, and on the other side, that none of the V missing pairs have domination; more succinctly, $H \in [K, K+V]$. Hence the confidence interval

$$\left[\frac{K}{m} - z_{\alpha/2}\sqrt{\frac{p(1-p)}{m}}, \frac{K+V}{m} + z_{\alpha/2}\sqrt{\frac{p(1-p)}{m}}\right].$$

is at least a $(1-\alpha)\%$ confidence interval, for the procedure with deliberately missing data[10].

4.4. **Divide-and-conquer with a trivial second half.**

Ignoring the paradigm that divide-and-conquer should balance its tasks, in (27) a very useful choice is $b = 1$. Loosely speaking, it reduces the cost of waiting-to-get lucky from order $n^{3/4}$ to order $n^{1/4}$.

The analysis of the speedup, relative to waiting-to-get-lucky, is easy.

[9]The pairing on $\{1, 2, \ldots, 2m\}$ must be assigned *before* observing which v items are missing. Pairing up the missing partitions, in order to get $\lceil v/2 \rceil$ missing pairs, in *not valid*; see Remark 5.

[10] We feel compelled to speculate that many users of "confidence intervals" don't really care about the confidence, nor the width of the interval, but really rely on the center of the interval, which in the standard case is an unbiased estimator. Our interval has center $(K + V/2)/m$, and this value is not an unbiased estimator; indeed, anything based on K, including the natural $K/(m-V)$, is biased in an unknowable way.

Theorem 2. *The speedup of the $b = 1$ procedure above, relative to the waiting-to-get-lucky algorithm described in Theorem 1, is asymptotically $\sqrt{n}/c$, with $c = \pi/\sqrt{6}$. Equivalently, the acceptance cost is asymptotically $2\,n^{1/4}\,6^{3/4}/\pi$.*

Proof. Recall that $x = e^{-c/\sqrt{n}}$. From (10) and (29), the acceptance cost $1/C$ is given by $C = \mathbb{P}(T = n)/\max_k \mathbb{P}(Z_1 = k) = \mathbb{P}(T = n)/\mathbb{P}(Z_1 = 0) = \mathbb{P}(T = n)/(1 - x)$. The comparison algorithm, waiting-to-get-lucky, has acceptance cost $1/\mathbb{P}(T = n)$. The ratio simplifies to $1/(1 - x) \sim \sqrt{n}/c$. □

To review, stage 1 is to simulate $(Z_2, Z_3, \ldots, Z_n)$, and accept it with probability *proportional to* the chance that $Z_1 = n - (2Z_2 + \cdots + nZ_n)$; the speedup comes from the brilliant idea in [32]. In contrast, waiting-to-get-lucky can be viewed as simulating $(Z_2, Z_3, \ldots, Z_n)$ and then *simulating* Z_1 to see whether or not $Z_1 = n - (2Z_2 + \cdots + nZ_n)$.

4.5. Self-similar iterative divide-and-conquer: $p(z) = d(z)p(z^2)$.

The methods of Sections 4.2 – 4.4 have acceptance costs that go to infinity with n. We now demonstrate an iterative divide-and-conquer that has an asymptotically constant acceptance cost.

A well-known result in partition theory is

$$p(z) = \prod_i (1 - z^i)^{-1} = \left(\prod_i 1 + z^i\right)\left(\prod_i \frac{1}{1 - z^{2i}}\right) = d(z)p(z^2), \tag{30}$$

where $d(z) = \prod_i 1 + z^i$ is the generating function for the number of partitions with distinct parts, and $p(z^2)$ is the generating function for the number of partitions where each part has an even multiplicity. This can of course be iterated to, for example, $p(z) = d(z)d(z^2)p(z^4)$, etc., and this forms the basis for a recursive algorithm.

Recall from (19) in Section 4.2, that each $Z_i \equiv Z_i(x)$ is geometrically distributed with $P(Z_i \geq k) = x^{ik}$. The *parity bit* of Z_i, defined by

$$\epsilon_i = 1(Z_i \text{ is odd}),$$

is a Bernoulli random variable $\epsilon_i \equiv \epsilon_i(x)$, with

$$\mathbb{P}(\epsilon_i(x) = 1) = \frac{x^i}{1 + x^i}, \quad \mathbb{P}(\epsilon_i(x) = 0) = \frac{1}{1 + x^i}. \tag{31}$$

Furthermore, $(Z_i(x) - \epsilon_i)/2$ is geometrically distributed, as $Z_i(x^2)$, again in the notation (19), and $(Z_i(x) - \epsilon_i)/2$ is independent of ϵ_i. What we really use is the converse: with $\epsilon_i(x)$ as above, independent of $Z_i(x^2)$, the $Z_i(x)$ constructed as

$$Z_i(x) := \epsilon_i(x) + 2Z_i(x^2), \quad i = 1, 2, \ldots$$

indeed has the desired geometric distribution.

Theorem 3. *The asymptotic acceptance cost for one step of the iterative divide-and-conquer algorithm using* $A = (\epsilon_1(x), \epsilon_2(x), \ldots)$*,* $B = ((Z_1(x) - \epsilon_1)/2, (Z_2(x) - \epsilon_2)/2, \ldots)$*, is* $\sqrt{8}$*.*

Proof. The acceptance cost $1/C$ can be computed via (29) and (26), with

$$\begin{aligned} C &= \frac{\mathbb{P}_x(T = n)}{\max_k \mathbb{P}_{x^2}(T = \frac{n-k}{2})} = \frac{\mathbb{P}_x(T = n)}{\max_k \mathbb{P}_{x^2}(T = k)} \sim \frac{\frac{1}{\sqrt{2\pi}\,\sigma(x)}}{\frac{1}{\sqrt{2\pi}\,\sigma(x^2)}} \\ &\sim \frac{n^{-3/4}}{(n/4)^{-3/4}} = 4^{-3/4} = 8^{-1/2}. \end{aligned}$$

□

Here is an informal discussion of the full algorithm. First, propose A until getting acceptance, then, since the B task is to find a uniformly chosen partition of a smaller integer, iterate to finish up. In effect, the iterative algorithm is to determine the $(Z_1, Z_2, \ldots, Z_n)$ conditional on $Z_1 + 2Z_2 + \ldots = n$, by finding the binary expansions: first the 1s bits of all the Z_is, then the 2s bits, then the 4s bits, and so on.

With a little more detail: to start, with $A = (\epsilon_1(x), \epsilon_2(x), \ldots)$ and $T_A = \sum_i i\,\epsilon_i$, we have $\mathbb{E}\,T_A = \sum_{i=1}^n \frac{ix^i}{1+x^i} \sim n/2$, and it can be shown that, even after conditioning on acceptance, the distribution of T_A is concentrated around $n/2$. Since $B = ((Z_1(x)-\epsilon_1)/2, (Z_2(x)-\epsilon_2)/2, \ldots)$ is equal in distribution to $(Z_1(x^2), Z_2(x^2), \ldots)$, and target $n' = (n - T_A)/2$, we see that the B phase is to find a partition of the integer n', uniform over the $p_{n'}$ possibilities. In carrying out the B task we simply use $x(n')$ as the parameter, but the choice $(x(n))^2$ would also work.

4.5.1. *Exploiting a parity constraint.* Theorem 3 states that the asymptotic acceptance cost for proposals of $A = (\epsilon_1(x), \epsilon_2(x), \ldots)$ is $2\sqrt{2}$, and this already takes into account an obvious lower bound of 2, since the parity of $T_A = \epsilon_1 + \epsilon_2 + \cdots + \epsilon_n$ is nearly equally distributed over {odd,even}, and rejection is guaranteed if T_A does not have the same parity as n. An additional speedup is attainable by moving ϵ_1 from the A side to the B side: instead of simulating ϵ_1, there now will be a trivial task, just as there was Z_1 in the "$b = 1$" procedure of Section 4.4. That is, we switch to $A = (\epsilon_2(x), \epsilon_3(x), \ldots)$ and $B = (\epsilon_1(x), (Z_1(x) - \epsilon_1)/2, (Z_2(x) - \epsilon_2)/2, \ldots)$; the parity of the new T_A dictates, deterministically, the value of the first component of B, under the conditioning on $h(a, B) = 1$. The rejection probabilities for a proposed A are like those in Theorem 3, but with an additional factor of $1/(1+x^1)$ or $x/(1+x^1)$, depending on the parity of $n+\epsilon_2+\cdots+\epsilon_n$. Since $x=x(n)\to 1$ as $n\to\infty$, these two factors both tend to $1/2$, so the constant C as determined by (29) becomes, asymptotically, twice as large.

Theorem 4. *The asymptotic acceptance cost for one step of the iterative divide-and-conquer algorithm using* $A=(\epsilon_2(x),\epsilon_3(x),\ldots)$ *and* $B=(\epsilon_1(x),(Z_1(x)-\epsilon_1)/2,(Z_2(x)-\epsilon_2)/2,\ldots)$ *is* $\sqrt{2}$.

Proof. The acceptance cost $1/C$ can be computed, as in the proof of Theorem 3, with the only change being that in the display for computing C, the expression under the $\max_k$, which was $\mathbb{P}(T(x^2)=(n-k)/2)$ changes to

$$\mathbb{P}\left(2|n-k+\epsilon_1(x) \text{ and } T(x^2)=\frac{n-k}{2}\right)$$
$$=\mathbb{P}(2|n-k+\epsilon_1(x))\times\mathbb{P}\left(T(x^2)=\frac{n-k}{2}\right)\sim\frac{1}{2}\mathbb{P}\left(T(x^2)=\frac{n-k}{2}\right).$$

□

4.5.2. *The overall cost of the main problem and all its subproblems.* Informally, for the algorithm in the previous section, the main problem has size n and acceptance cost $\sqrt{2}$, applied to a proposal cost asymptotic to $c_0\sqrt{n}$, for a net cost $\sqrt{2}\,c_0\sqrt{n}$. The first subproblem has random size, concentrated around $n/4$, and hence half the cost of the main problem. The sum of a geometric series with ratio $1/2$ is twice the first term, so the net cost of the main problem and all subproblems combined is $2\sqrt{2}\,c_0\sqrt{n}$.

In framing a Theorem to describe this, we try to allow for a variety of costing schemes. We believe that the first sentence in the hypotheses of Theorem 5 is valid, with $\theta=1/2$, for the scheme of Remark 4. The second sentence, about costs of tasks other than proposals, is trivially true for the scheme of Remark 4, but may indeed be false, in costing schemes which assign a cost to memory allocation, and communication.

Theorem 5. Assume *that the cost* $C(n)$ *to propose the* $A=(\epsilon_2(x),\epsilon_3(x),\ldots)$ *in the first step of the algorithm of Section 4.5.1, is given by a deterministic function with* $C(n)\sim c_0n^\theta$ *for some positive constant* c_0 *and constant* $\theta\geq 1/2$, *or even more generally,* $C(n)=n^\theta$ *times a slowly varying function of* n. *Assume that the cost of all steps of the algorithm, other than making proposals,*[11] *is relatively negligible, i.e.,*

[11]such as the arithmetic to compute acceptance/rejection thresholds, the generation of the random numbers used in those acceptance/rejection steps, and merging the accepted proposals in to a single partition

$o(C(n))$. Then, the asymptotic cost of the entire algorithm is

$$\frac{1}{1-1/4^{\theta}}\sqrt{2}C(n) \leq 2\sqrt{2}\,C(n).$$

Proof. The key place to be careful is in the distinction between the distribution of a proposed $A = (\epsilon_2(x), \epsilon_3(x), \ldots)$, and the distribution after rejection/acceptance. For proposals, in which the ϵ_i are mutually independent, with $T_A := \sum_2^n i\,\epsilon_i(x)$, with $x = x(n)$ from (23), calculation gives $\mathbb{E}\,T_A \sim n/2$ and $\text{Var}(T_A) \sim (1/c)n^{3/2}$. Chebyshev's inequality for being at least k standard deviations away from the mean, to be used with $k = k(n) = o(n^{1/4})$, and $k \to \infty$, gives $\mathbb{P}(|T_A - \mathbb{E}\,T_A| > k\,\text{SD}(T_A)) \leq 1/k^2$.

Now consider the good event G that a proposed A is accepted; conditional on G, the ϵ_i are no longer mutually independent. But the upper bound from Chebyshev is robust, with $\mathbb{P}(|T_A - \mathbb{E}\,T_A| > k\,\text{SD}(T_A)|G) \leq 1/(k^2\,\mathbb{P}(G))$. Since $\mathbb{P}(G)$ is bounded away from zero, by Theorem 4, we still have an upper bound which tends to zero, and shows that $(n - T_A)/2$, divided by n, converges in probability to $1/4$.

Write $N_i \equiv N_i(n)$ for the random size of the subproblem at stage i, starting from $N_0(n) = n$. The previous paragraph showed that for $i = 0$, $N_{i+1}(n)/N_i(n) \to 1/4$, where the convergence is *convergence in probability*, and the result extends automatically to each fixed $i = 0, 1, 2, \ldots$. We have deterministically that $N_{i+1}/N_i \leq 1/2$, so in particular $N_i > 0$ implies $N_{i+1} < N_i$. Set $C(0) = 0$, redefining this value if needed, so that the costs of all proposals is exactly the random

$$S(n) := \sum_{i\geq 0} C(N_i(n)).$$

It is then routine analysis to use the hypothesis that $C(n)$ is regularly varying, to conclude that $S(n)/C(n) \to 1/(1-4^{-\theta})$, where again, the convergence is *convergence in probability.* The deterministic bound $N_{i+1}(n)/N_i(n) \leq 1/2$ implies that the random variables $S(n)/C(n)$ are *bounded*, so it also follows that $\mathbb{E}\,S(n)/C(n) \to 1/(1-4^{-\theta})$. □

4.5.3. *A variation based on $p(z) = p_{odd}(z)\,p(z^2)$.* With

$$p_{odd}(z) := \prod_{i \text{ odd}} (1-z^i)^{-1},$$

Euler's identity $d(z) = p_{odd}(z)$ suggests a variation on the algorithm of section 4.5. It is arguable, whether the original algorithm, based on $p(z) = d(z)\,p(z^2)$, and the variant, based on $p(z) = p_{odd}(z)\,p(z^2)$, are genuinely different.

Arguing the variant algorithm is different: the initial proposal is $A = (Z_1(x), Z_3(x), Z_5(x), \ldots)$. Upon acceptance, we have determined $(C_1(\lambda), C_3(\lambda), C_5(\lambda), \ldots)$, where λ is the partition of n that the full iterative algorithm will determine, and $C_i(\lambda)$ is the number of parts of size i in λ. The B task will find $(C_2(\lambda), C_4(\lambda), C_6(\lambda), \ldots)$ by iterating the divide-and-conquer idea, so that the second time through the A procedure determines $(C_2(\lambda), C_6(\lambda), C_{10}(\lambda), \ldots)$, and the third time through the A procedure determines $(C_4(\lambda), C_{12}(\lambda), C_{20}(\lambda), \ldots)$, and so on.

Arguing that the variant algorithm is *essentially* the same: just as in Euler's bijective proof that $p_{odd}(z) = d(z)$, the original algorithm had a proposal $A = (\epsilon_1(x), \epsilon_2(x), \ldots)$, which can be used to construct the proposal $(Z_1(x), Z_3(x), Z_5(x), \ldots)$ for the variant algorithm. That is, one can check that starting with independent $\epsilon(i,x) \equiv \epsilon_i(x)$ given by (31), for $j = 1, 3, 5, \ldots$, $Z_j := \sum_{m \geq 0} \epsilon(j\,2^m, x)\; 2^m$ indeed has the distribution of $Z_j(x)$ specified by (19), with $Z_1, Z_3, \ldots$ independent. And conversely, one can check that starting with the independent geometrically distributed $Z_1(x), Z_3(x), \ldots$, taking base 2 expansions yields mutually independent $\epsilon_1(x), \epsilon_2(x), \ldots$ with the Bernoulli distributions specified by (31). Hence one *could* program the two algorithms so that they are coupled: starting with the same seed, they would produce the same sequence of colors T_A for the initial proposal, and the same count of rejections before the acceptance for the first time through the A procedure, with same T_A for that first acceptance, and so on, including the same number of iterations before finishing. Under this coupling, the original algorithm produces a partition μ of n, the variant algorithm produces a partition λ of n — and we have implicitly defined the deterministic bijection f with $\lambda = f(\mu)$.

Back to arguing that the algorithms are different: we believe that the coupling described in the preceding paragraph supplies rigorous proofs for the analogs of Theorems 3 and 4. For Theorem 5 however, one should also consider the computational cost of Euler's bijection, for various costing schemes, and we propose the following analog, for the variant based on $p(z) = p_{odd}(z)\,p(z^2)$, combined with the trick of moving $\epsilon_1(x)$ from the A side to the B side, as in Section 4.5.1:

Theorem 6. Assume *that the cost $D(n)$ to propose $(Z_1(x), Z_2(x), \ldots, Z_n(x))$, with $x = x(n)$, satisfies $D(n) = n^\theta$ times a slowly varying function of n. Assume* also *that the cost $D_A(n)$ to propose $A = (Z_1(x) - \epsilon_1(x), Z_3(x), Z_5(x), \ldots)$ satisfies $D_A(n) \sim D(n)/2$.*

Then, the asymptotic cost of the entire algorithm is

$$\frac{1}{1-1/4^{\theta}}\sqrt{2}\,D(n)/2 \le \sqrt{2}\,D(n).$$

Proof. Essentially the same as the proof of Theorem 5. □

It is plausible that the cost function $C(n)$ from Theorem 5 and the the cost function $D(n)$ from Theorem 6 are related by $C(n) \sim D(n)$; note that this depends on the choice of costing scheme, essentially asking whether or not the algorithmic cost of carrying out Euler's odd-distinct bijection is negligible.

Another argument that the two algorithms are different arises from the *completely artificial* example at the end of Section 5.4.

5. For integer partitions: review, method of choice

In Section 4, five methods were presented for the simulation of partitions of the integer n. Now we review the costs of running these algorithms, taking into account the size of n, the number m of samples to be delivered, and so on. We also consider alternate simulation tasks involving integer partitions with restrictions on the parts to see how the methods adapt.

5.1. **Method of choice for unrestricted partitions.**

If one is interested in generating just a few partitions of a moderately sized n, then the waiting-to-get-lucky method, with a dumb "time n" method of proposal for $(Z_1, \ldots, Z_n)$, is very easy to program. The overall runtime is order of $n^{7/4}$ — a factor of n to make a proposal, and a factor of $n^{3/4}$ for waiting to get lucky.[12] For example, in Matlab® [24]

```
n=100; logx=-sqrt(6*n)/pi; s=0;
while s~=n,
    Z=floor(log(rand(n,1))./(1:n)'.*logx);
    s=(1:n)*Z;
end
```

runs on a common desktop computer[13] at around 600 partitions of $n = 100$ per second; with $n = 1000$ the same runs at about 20 partitions per second, and at $n = 10,000$ takes about 2 seconds per partition.

[12]A smarter "time $\sqrt{n}$" method of proposal for $(Z_1, \ldots, Z_n)$ is described in Secton 5.2. It is harder to program, but gets the overall runtime down to order of $n^{5/4}$.

[13]Macintosh iMac, 3.06 Ghz, 4 GB RAM

The table method is by far the fastest method, if one is interested in generating many samples, and the table of size n^2 floating point numbers fits in random access memory. For example, for $n = 10,000$ the same computer as above takes 5 seconds to generate the table — a one time cost, and then finds 40 partitions per second. At $n = 15,000$, the same computer takes 28 seconds to generate the table, and then finds 25 partitions per second. But at $n = 19,000$, the computer freezes, as too much memory was requested.

The divide-and-conquer methods of Sections 4.3 and 4.4, using the small versus large division of (27), offer a large speedup over waiting-to-get-lucky, but only for case $b = 1$, with its trivial second half, can we analyze the speedup — the $\sqrt{n}/c$ factor in Theorem 2.

The divide-and-conquer method based on $p(z) = d(z)p(z^2)$ is unbeatable for large n. Regardless of the manner of costing, be it only counting random bits used, or uniform costing, or logarithmic (bitop) costing, the cost to find a random partition of n must be asymptotically at least as large as the time to propose $(Z_1, \ldots, Z_n)$ for a random partition of a random number around n. The entire divide-and-conquer algorithm of Theorem 6, compared with just proposing $(Z_1, \ldots, Z_n)$, has asymptotically an extra cost factor of $\sqrt{2}$. So the claim of *unbeatable* at the start of this paragraph really means: asymptotically unbeatable by anything more than a factor of $\sqrt{2}$.

5.2. **Complexity considerations.** At the end of Section 2.2 we note that in the general view of probabilistic divide-and-conquer algorithms, a key consideration is computability of the acceptance threshold $t(a)$. The case of integer partitions, using any of the divide-and-conquer algorithms of Section 4.5, is perhaps exceptionally easy, in that computing the acceptance threshold is essentially the same as evaluating p_m, an extremely well-studied task. For $m > 10^4$ a *single term* of the Hardy-Ramanujan asymptotic series suffices to evaluate p_m with relative error less than 10^{-16}; see Lehmer [21, 22].[14] This single term is

$$hr_1(n) := \frac{\exp(y)}{4\sqrt{3}(n - \frac{1}{24})} \left(1 - \frac{1}{y}\right), \text{ where } y = 2c\sqrt{n - \frac{1}{24}},$$

and numerical tabulation[15], together with Lehmer's guarantee, shows that

$$|\ln p_n - \ln hr_1(n)| < 10^{-16} \text{ for all } n \geq 810.$$

[14]We thank David Moews for bringing these papers to our attention.

[15]done in Mathematica® 8.

Is floating point accuracy sufficient, in the context of computing an acceptance threshold $t(a)$? There is a very concrete answer, based on [20]. First, as in Lemma 7.14 in [2], given $p \in (0,1)$, a p-coin can be tossed using a random number of fair coin tosses; the expected number is exactly 2, unless p is a kth level dyadic rational, i.e., $p = i/2^k$ with odd i, in which case the expected number is $2 - 2^{1-k}$. The proof is by consideration of say $B, B_1, B_2, \ldots$ iid with $\mathbb{P}(B = 0) = \mathbb{P}(B = 1) = 1/2$; after r tosses we have determined the first r bits of the binary expansion of a random number U which is uniformly distributed in $(0,1)$, and the usual simulation recipe is that a p-coin is the indicator $1(U < p)$. Unless $\lfloor 2^r\, p \rfloor = \lfloor 2^r\, U \rfloor$, the first r fair coin tosses will have determined the value of the p-coin. Exchanging the roles of U and p, we see that number of bits of precision read off of p is, on average, 2, and exceeds r with probability 2^{-r}. If a floating point number delivers 50 bits of precision, the chance of needing more precision is 2^{-50}, per evaluation of an indicator of the form $1(U < p)$. Our divide-and-conquer doesn't require very many acceptance/rejection decisions; for example, with $n = 2^{60}$, there are about 30 iterations of the algorithm in Theorem 4, each involving on average about $\sqrt{2}$ acceptance/rejection decisions, according to Theorem 3. So one might program the algorithm to deliver exact results; most of the time determining acceptance thresholds $p = t(a)$ in floating point arithmetic, but keeping track of whether more bits of p are needed. On the unlikely event, of probability around $30 \times \sqrt{2}/2^{50} < 4 \times 10^{-14}$, that more precision is needed, the program demands a more accurate calculation of $t(a)$. This would be far more efficient than using extended integer arithmetic to calculate values of p_n exactly.

Another place to consider the use of floating point arithmetic is in proposing the vector $(Z_1(x), \ldots, Z_n(x))$. If one call to the random number generator suffices to find the next arrival in a rate 1 Poisson process, we have an algorithm using $O(\sqrt{n})$ calls, which can propose the entire vector $(Z_1, Z_2, \ldots)$, using $x = x(n)$ from (23). The proposal algorithm is based on a compound Poisson representation of geometric distributions, and is similar to a coupling used in [3], Section 3.4.1. The supporting calculation here is that, with $x = x(n) = \exp(-c/\sqrt{n})$ and $c = \pi/\sqrt{6}$,

$$s(n) := \sum_{i,j \geq 1} \frac{x^{ij}}{j} = \sum_j \frac{1}{j^2} \frac{j x^j}{1 - x^j} \leq \frac{\pi^2}{6} \frac{x}{1-x} \sim c\sqrt{n}.$$

The algorithm constructs the rate 1 Poisson process on $(0, s(n))$, with the full interval partitioned into subintervals of lengths x^{ij}/j. With $Y_{i,j}$

equal to the number of arrivals in the subinterval of length x^{ij}/j, the $Y_{i,j}$ are mutually independent, Poisson distributed, and $Z_i := \sum jY_{i,j}$ constructs the desired mutually independent geometrically distributed $Z_1(x), Z_2(x), \ldots$.

Once again, suppose we want to guarantee *exact* simulation of a proposal $(Z_1(x), \ldots, Z_n(x))$. In the compound Poisson process with $s(n)$ arrivals expected, we need to assign *exactly*, for each arrival, say at a random time R, the corresponding index (i, j), such the partial sum for $s(n)$ up to, but excluding the ij term, is less than R, but the partial sum, including the ij term, is greater than or equal to R. Based on an entropy result from Knuth-Yao [20], a crucial quantity is

$$h(n) := \sum_{i,j\geq 1} \frac{x^{ij}}{j} \log \frac{j}{x^{ij}} \leq (c - \zeta'(2)/c)\ \sqrt{n}.$$

An exact simulation of the Poisson process, assigning ij labels to each arrival, can be done[16] with $O(s(n) + h(n))$ genuine random bits, and the bounds for $s(n)$ and $h(n)$ show that this is $O(\sqrt{n})$.

The costing scheme which counts only the expected number of random bits needed is clearly inadequate. Consider the impractical algorithm: list all p_n partitions, for example in lexicographic order. Use $\lceil \log_2 p_n \rceil$ random bits to choose an integer I uniformly from $\{1, 2 \ldots, p_n\}$. Report the Ith partition of n. If one costs only by the number of random bits needed, the algorithm just described is *strictly* unbeatable!

5.3. **Partitions with restrictions.** As with unrestricted partitions, if n is moderate and a recursive formula exists, analogous to that of Section 4.1, then the table method is the most rapid, and divide-and-conquer is not needed. However, the requirement of random access storage of size n^2 is a severe limitation.

The self-similar iterative divide-and-conquer method of Section 4.5 is nearly unbeatable for large n, for ordinary partitions. There are many classes of partitions with restrictions that iterate nicely, and should be susceptible to a corresponding iterative divide-and-conquer algorithm. Some of these classes, with their self-similar divisions, are

(1) distinct parts, $d(z) = d_{odd}(z)d(z^2)$;

[16]on each interval $(m-1, m]$ for $m = 1$ to $\lceil s(n) \rceil$, perform an exact simulation of the number of arrivals, which is distributed according to the Poisson distribution with mean 1. For each arrival on $(m-1, m]$, there is a discrete distribution described by those partition points lying in $(m-1, m)$, together with the endpoints $m-1$ and m; calling the corresponding random variable X_m, the sum of the base 2 entropies satisfies $\sum h(X_m) \leq h(n) + s(n)$, since each extra subdivision of one of the original subintervals of length x^{ij}/j adds at most one bit of entropy.

(2) odd parts, $p_{odd}(z) = d_{odd}(z)p_{odd}(z^2)$;
(3) distinct odd parts, $d_{odd}(z) = d_*(z)d_{odd}(z^3)$.

Here $d_*(z) = (1+z)(1+z^5)(1+z^7)(1+z^{11})\cdots$ represents distinct parts $\equiv \pm 1 \mod 6$. Other recurrences are discussed in [18, 27, 31], and the standard text on partitions [1].

It is not easy to come up with examples where the optimal divide-and-conquer is like that in Section 4.3, based on small parts versus large parts. One suggestion is partitions with all parts prime; there should be a large range of n for which table methods are ruled out by the memory requirement, while the n memory, $b \times n$ computational time to calculate rejection probabilities is not prohibitive. Another suggestion is partitions with a restriction on the multiplicity, for example, a part of size i can occur at most $f(i)$ times — with f sufficiently complicated as to rule out iterative formulas such as those in the preceding parapgraph. Another family is, for $d = 2, 3, \ldots$, partitions where all parts differ by at least d.

5.4. An eye for gathering statistics.

The underlying motivation for sampling a random element S, m times under some given distribution μ, might be to produce a statistic $h(S_1, S_2, \ldots, S_m)$ based on that sample. Conceivably, the deterministic function h might only look at some part of the data, so that there is a divide-and-conquer scheme in which $S = (X, Y)$, using the notation of Lemma 1, and $h(S_1, S_2, \ldots, S_m) = g(X_1, X_2, \ldots, X_m)$ for a deterministic g. In this case, in the notation of Section 2.1, one need only carry out Stage 1.

Here is a completely artificial example, motivated by the question of dominance, discussed in Section 4.3.3. For a partition λ of n, define $f(\lambda)$ to be the partition, having distinct parts, in which i is a part of $f(\lambda)$ if and only if i has *odd* multiplicity as a part of λ. Now suppose that one needs to approximate, for say $n = 10^9$, $\pi_n^{\text{parity}} = \mathbb{P}(f(\lambda)$ dominates $f(\mu))$, choosing (λ, μ) uniformly over the $(p_n)^2$ possible pairs of partitions of n.

6. For comparison: opportunistic divide-and-conquer with mix-and-match

Recall the setup of (1) – (4). In the third and fourth paragraphs of Section 1.2, we describe our starting point, a motivation stemming from mix-and-match, and admit that we can find no way to carry out "opportunistic" mix-and-match, to get a perfect sample. Here, we point out that nonetheless, the natural opportunistic procedure does supply a *consistent* estimator.

Take a deterministic design: for integers $m_1, m_2 \geq 1$, let $A_1, \ldots, A_{m_1}$ be distributed according to the distribution of A given in (1), and let $B_1, \ldots, B_{m_2}$ be distributed according to the distribution of B, with these $m_1 + m_2$ sample values being mutually independent. The **opportunistic** observations, under a *take-all-you-can-get* strategy, are all the pairs (A_i, B_j) for which $h(A_i, B_j) = 1$. To use these in an estimator, one would naturally count the available pairs, via

$$W = W(m_1, m_2) = \sum_{1 \leq i \leq m_1, 1 \leq j \leq m_2} h(A_i, B_j), \tag{32}$$

and for a deterministic function $g : \operatorname{support} h \subset \mathcal{A} \times \mathcal{B} \to \mathbb{R}$, form the total score from these pairs, say

$$G \equiv G(m_1, m_2) = \sum_{1 \leq i \leq m_1, 1 \leq j \leq m_2} g(A_i, B_j)\, h(A_i, B_j). \tag{33}$$

The natural estimator is the observed average score per pair, G/W. Unfortunately, this is not unbiased. However, it is consistent, i.e., as $m_1, m_2 \to \infty$, $G/W \to \mathbb{E}\, g(S)$ in probability.

Theorem 7. *For bounded g, we get an unbiased estimate of $p\, \mathbb{E}\, g(S)$ from $G/(m_1 m_2)$. Furthermore, as $m_1, m_2 \to \infty$ $G(m_1, m_2)/(m_1 m_2) \to p\, \mathbb{E}\, g(S)$, where the convergence is convergence in probability.*

Proof. To show unbiased: for each i, j we have, since h is an indicator, with $\mathbb{E}\, h(A_i, B_j) = p$,

$$\mathbb{E}\, g(A_i, B_j)\, h(A_i, B_j) = \mathbb{E}\left(g(A_i, B_j)\, | h(A_i, B_j) = 1\right) \times p.$$

According to (4), the conditional distribution of (A_i, B_j) given $h = 1$ is equal to the distribution of S, so the right side of the display above equals $p\, \mathbb{E}\, g(S)$.

For consistency: write $X_{ij} = g(A_i, B_j)\, h(A_i, B_j)$. In the variance-covariance expansion of $\operatorname{Var}(G)$, there are $m_1 m_2$ terms $\operatorname{cov}(X_{ij}, X_{i'j'})$ where both $i = i'$ and $j = j'$, $m_1 m_2 (m_2 - 1)$ terms where only $i = i'$, $m_1(m_1 - 1) m_2$ terms where only $j = j'$, and all other terms — involving four independent random variables $A_i, A_{i'}, B_j, B_{j'}$ — are zero. Hence $\operatorname{Var}(G/(m_1 m_2)) \to 0$, and Chebyshev's inequality implies the desired convergence in probability. □

Observe that in the special case $g = 1$, the random variable G is the count W, so Theorem 7 implies that $W/(m_1 m_2) \to p$. This shows that G/W is a consistent estimator of $\mathbb{E}\, g(S)$.

References

[1] George E. Andrews. *The Theory of Partitions*. Cambridge Mathematical Library, 1984.

[2] Sanjeev Arora and Boaz Barak. *Computational complexity*. Cambridge University Press, Cambridge, 2009. A modern approach.

[3] R. Arratia. On the amount of dependence in the prime factorization of a uniform random integer. In *Contemporary combinatorics*, volume 10 of *Bolyai Soc. Math. Stud.*, pages 29–91. János Bolyai Math. Soc., Budapest, 2002.

[4] Richard Arratia, Louis Gordon, and Michael S. Waterman. The Erdős-Rényi law in distribution, for coin tossing and sequence matching. *Annals of Statistics*, 18(2):529–570, 1990.

[5] Richard Arratia and Simon Tavaré. Independent process approximations for random combinatorial structures. *Advances in Mathematics*, 104:90–154, 1994.

[6] Richard Arratia and Michael S. Waterman. Critical phenomena in sequence matching. *Annals of Probability*, 13(4):1236–1249, 1985.

[7] James Antonio Bucklew. *Introduction to rare event simulation*. Springer Series in Statistics. Springer-Verlag, New York, 2004.

[8] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms*. MIT Press, 2001.

[9] Alain Denise and Paul Zimmermann. Uniform random generation of decomposable structures using floating-point arithmetic. *Theoret. Comput. Sci.*, 218(2):233–248, 1999. Caen '97.

[10] Persi Diaconis and Arun Ram. A probabilistic interpretation of the macdonald polynomials. *To Appear*, 2010.

[11] Philippe Duchon, Philippe Flajolet, Guy Louchard, and Gilles Schaeffer. Boltzmann samplers for the random generation of combinatorial structures. *Combin. Probab. Comput.*, 13(4-5):577–625, 2004.

[12] Bert Fristedt. The structure of random partitions of large integers. *Trans. Amer. Math. Soc.*, 337(2):7–3–735, 1993.

[13] Jason Fulman, Jan Saxl, and Pham Huu Tiep. Cycle indices for finite orthogonal groups of even characteristic. *Transactions of the American Mathematical Society*, 2011.

[14] G H Hardy and S Ramanujan. Asymptotic formulae in combinatory analysis. *Proc. London Math. Soc*, pages 75 – 115, 1918.

[15] Michael T. Heideman, Don H. Johnson, and C. Sidney Burrus. Gauss and the history of the fast Fourier transform. *Arch. Hist. Exact Sci.*, 34(3):265–277, 1985.

[16] A. Karatsuba and Yu. Ofman. Multiplication of many-digital numbers by automatic computers. *Proceedings of the USSR Academy of Sciences*, 145:293–294, 1962.

[17] A. A. Karatsuba. The complexity of computations. *Trudy Mat. Inst. Steklov.*, 211(Optim. Upr. i Differ. Uravn.):186–202, 1995.

[18] Charles Knessl and Joseph B. Keller. Partition asymptotics from recursion equations. *SIAM J. Appl. Math.*, 50(2):323–338, 1990.

[19] Donald E. Knuth. *The art of computer programming. Vol. 4, Fasc. 3*. Addison-Wesley, Upper Saddle River, NJ, 2005. Generating all combinations and partitions.

[20] Donald E. Knuth and Andrew C. Yao. The complexity of nonuniform random number generation. In *Algorithms and complexity (Proc. Sympos., Carnegie-Mellon Univ., Pittsburgh, Pa., 1976)*, pages 357–428. Academic Press, New York, 1976.

[21] D. H. Lehmer. On the series for the partition function. *Trans. Amer. Math. Soc.*, 43(2):271–295, 1938.

[22] D. H. Lehmer. On the remainders and convergence of the series for the partition function. *Trans. Amer. Math. Soc.*, 46:362–373, 1939.

[23] Mathematica. *Mathematica Edition: Version 8.0.* Wolfram Research, Inc., Champaign, IL, 2010.

[24] MATLAB. *version 7.12.0.635 (R2011a).* The MathWorks Inc., Natick, Massachusetts, 2011.

[25] A. Nijenhuis and H. S. Wilf. A method and two algorithms on the theory of partitions. *J. Combinatorial Theory Ser. A*, 18:219–222, 1975.

[26] Albert Nijenhuis and Herbert S. Wilf. *Combinatorial algorithms.* Academic Press Inc. [Harcourt Brace Jovanovich Publishers], New York, second edition, 1978. For computers and calculators, Computer Science and Applied Mathematics.

[27] Igor Pak. Partition bijections, a survey. *Ramanujan J.*, 12(1):5–75, 2006.

[28] Boris Pittel. On a likely shape of the random Ferrers diagram. *Adv. in Appl. Math.*, 18(4):432–488, 1997.

[29] Boris Pittel. Confirming two conjectures about the integer partitions. *J. Combin. Theory Ser. A*, 88(1):123–135, 1999.

[30] James Gary Propp and David Bruce Wilson. Exact sampling with coupled Markov chains and applications to statistical mechanics. In *Proceedings of the Seventh International Conference on Random Structures and Algorithms (Atlanta, GA, 1995)*, volume 9, pages 223–252, 1996.

[31] Jeffrey B. Remmel. Bijective proofs of some classical partition identities. *J. Combin. Theory Ser. A*, 33(3):273–286, 1982.

[32] John von Neumann. Various techniques used in connection with random digits. monte carlo methods. *National Bureau of Standards*, pages 36–38, 1951.

(Richard Arratia) Department of Mathematics, University of Southern California, Los Angeles CA 90089.
E-mail address: `rarratia@math.usc.edu`

(Stephen DeSalvo) Department of Mathematics, University of Southern California, Los Angeles CA 90089.
E-mail address: `stephen.desalvo@gmail.com`